Understanding Federated Learning via Client-Level Influence Measurement

ANONYMOUS AUTHOR(S)

Federated learning allows mobile clients to jointly train a global model without sending their private data to a central server. Despite that extensive works have studied the performance guarantee of the global model, it is still unclear how each individual client influences the collaborative training process. In this work, we defined a novel notion, called *Fed-Influence*, to quantify this influence in terms of model parameter, and proposed an effective and efficient estimation algorithm. In particular, our design satisfies several desirable properties: (1) it requires neither retraining nor retracing, adding only linear computational overhead to clients and the server; (2) it strictly maintains the tenet of federated learning, without revealing any client's local data; and (3) it works well on both convex and non-convex loss functions and does not require the final model to be optimal. Empirical results on a synthetic dataset and the FEMNIST dataset show that our estimation method can approximate Fed-Influence with small bias. Further, we demonstrated that influence measurement gives us a reasonable evaluation of clients, which enables a client-level model debugging.

CCS Concepts: • **Computing methodologies → Distributed artificial intelligence**; • **Social and professional topics → Quality assurance**.

Additional Key Words and Phrases: federated learning; influence measurement

ACM Reference Format:
Anonymous Author(s). XXXX. Understanding Federated Learning via Client-Level Influence M easurement. 1, 1 (October XXXX), 21 pages. https://doi.org/XX.XXXX/XXXXXX.XXXXXXX

1 INTRODUCTION

Federated learning ingeniously leverages a large amount of valuable data in a distributed manner, while mitigating systemic privacy risks [11, 19]. In such a setting, the training data are stored on multiple decentralized clients, who share a common global model and train it collaboratively. There is a central server that orchestrates the whole process round by round. In each round, the server collects local models from some eligible clients and using them to update the global model.

In this paper, we consider a significantly important but overlooked problem in federated learning: *how does each client influence the global model?* Finding the answer to this question is meaningful in several aspects. On the one hand, it provides insights into the roles of individual clients in federated learning, and informs us whether the existence of a certain client benefits the global model's training. A more fine-grained understanding of clients' influence allows us to quantify the performance of clients in federated learning, which is critical to a fair credit/reward allocation. On the other hand, client-level influence measurement can also be used to debug federated learning. In a centralized setting, modelers can directly check the data when the model is misbehaving, while in federated learning they suffer from a lack of data inspection. Thus, a good understanding of clients' influence further facilitates the online removal of low-quality clients or dynamically requires low-quality clients to check their local data, thereby improving model performance. All of these are important to the interpretability and robustness of federated learning, and also help sustain long-term user participation.

There already exists a classical statistics notion of “influence" in centralized learning, which evaluates the effect that the absence of an individual sample has on a model. To measure this influence, one should conduct a leave-one-out test [3]: retrain the model over the training set with one certain sample removed, and compare this model with that trained on the full dataset. A notion from robust statistics called “influence function" [2, 8, 10], was introduced to avoid retraining by measuring the change in the model caused by slightly changing the weight of one sample and using quadratic approximation combined with a Newton step [4]. Koh and Liang [15] leveraged influence function in modern machine learning settings and developed an efficient and simple implementation using second-order optimization techniques. Hara et al. [9] went beyond convexity and optimality, which are two important assumptions in [15]. Koh et al. [14] studied the effects of removing a group of data points, which is analogous to removing a client that holds a subset of training data in federated learning, except that their study is still in a centralized setting. Similar work by Khanna et al. [13] applied Fisher kernels along with Sequential Bayesian Quadrature (SBQ) to identify a subset of training examples that are most responsible for a given set of predictions and recovered [15] as a special case.

The existing works above focused on centralized learning, and we are the first to consider a similar problem, the influence of individual clients, under a brand new framework, namely federated learning. There are some essential differences between centralized learning and federated learning which raise several design challenges: (1) The server in centralized learning, as the influence evaluator, has the full control over the considered sampling data, while in federated learning, the server would not be able to access clients' data because of the privacy requirement; (2) the clients in federated learning may not always be available, due to the unreliable network connection. This implies that the server cannot communicate with a certain client at any desired time; and (3) the computing resources of mobile clients are limited, for which reason we should not bring too many additional computational burdens to clients when measuring their influence.

Due to the first difference, sample-level influence in centralized learning cannot be applied to measuring the influence of individual clients in federated learning. In this work, we turn to client-level influence measurement by investigating the effect of removing a client. In addition, works in centralized learning mainly focus on the influence on a model's testing loss, while we consider the influence on the parameter of the global model for the following two reasons: (1) In centralized learning, to cut down the time complexity, some techniques can be applied to obtain influence on loss without computing that on parameter [9]. However, in federated learning, computing influence on parameter is unavoidable because of the second and third differences mentioned earlier. Please refer to Section 7 for more detailed reasoning; and (2) influence on parameter is more fundamental and powerful than that on loss. With the knowledge of influence on parameter, we can easily derive the influence of individual clients in terms of various metrics for evaluating models, such as loss, accuracy, precision, etc.

We summarize our key contributions in this work as follows. (1) To the best of our knowledge, we are the first to consider client-level influence in federated learning; (2) we propose a basic estimator for individual clients' influence on model parameter by leveraging the relationship between the global models in two consecutive communication rounds. Guided by the error analysis, we extend the basic design to support both convex and non-convex loss functions. We also develop an efficient implementation, bringing only slight communication and computation overhead to the server and the clients (in fact no extra computation overhead for the client); and (3) empirical studies on a synthetic dataset and the FEMNIST dataset [1] demonstrate the effectiveness of our method. The estimation error observed in experiments indicates both the necessity and accuracy of our method.

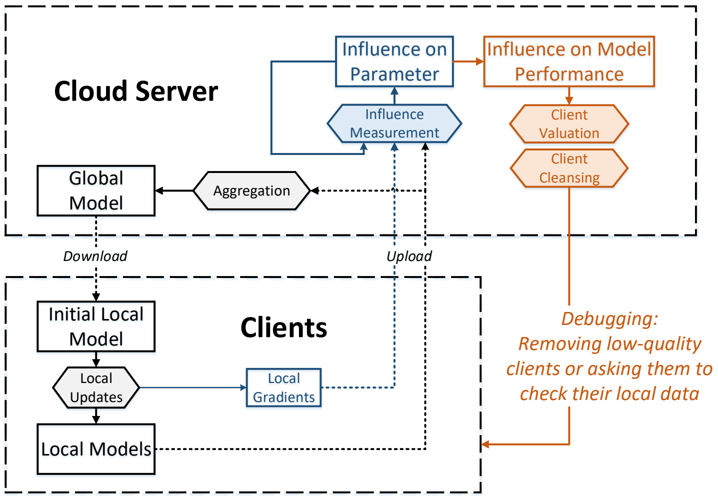


Fig. 1. An overview of our work.

Based on the influence on parameter, we further derive the influence on model performance and observe that it was well estimated. In particular, the Pearson correlation coefficient between the estimated influence on loss and the ground truth achieves 0.62 in the most difficult setting. We also leverage influence on model performance for client valuation and client cleansing, which achieves a 0.53% increase in accuracy by removing 5% of the clients in the most difficult setting.

Fig. 1 gives an overview of the paper structure, which is composed of three parts. The black part depicts the original framework of federated learning introduced in Subsection 2.1. The blue part shows our design in Sections 3 and 4 to measure client-level influence, which works parallel with the model training. The orange part illustrates how influence measurement can benefit model performance, and the detailed results are presented in Section 6.

2 PROBLEM FORMULATION

2.1 Federated Learning

We first introduce some necessary notations. C denotes the set of all the clients. $\mathcal{D}^k$ denotes the local dataset of client $k \in C$ with n_k samples. $\mathcal{D} = \bigcup_{k \in C} \mathcal{D}^k$ is the full training set. For an arbitrary set of clients C', $N(C') = \sum_{k \in C'} n_k$ denotes the total size of these clients' datasets. $\mathcal{L}(\mathbf{w}, z)$ denotes the loss function over a model $\mathbf{w}$ and a sample z. In addition, $\mathcal{L}(\mathbf{w}, \mathcal{D}^k) = \frac{1}{n_k} \sum_{z \in \mathcal{D}^k} \mathcal{L}(\mathbf{w}, z)$ denotes the empirical loss over a model $\mathbf{w}$ and $\mathcal{D}^k$. Then, we consider the following optimization task of federated learning:

$$\min_{\mathbf{w} \in \mathbb{R}^p} \left\{ \mathcal{L}(\mathbf{w}, \mathcal{D}) = \sum_{k \in C} \frac{n_k}{N(C)} \mathcal{L}(\mathbf{w}, \mathcal{D}^k) \right\}, \tag{1}$$

where the global loss function $\mathcal{L}(\mathbf{w}, \mathcal{D})$ is the weighted average of the local functions $\mathcal{L}(\mathbf{w}, \mathcal{D}^k)$ with the weight of each client proportioning to the size of its local dataset. In this work, we consider a standard algorithm, federated averaging (FedAvg) [19], to solve Equation 1. Although there are some other variants, such as FedBoost [7], FedNova [21], FetchSGD [20], FedProx [16], and SCAFFOLD [12], FedAvg is the first and the most widely used one. As a result, we see FedAvg as our basic block, which executes as follows. In the initial stage, the server randomly initializes a global model $\mathbf{w}_0$. Then, the training is orchestrated by repeating the following two steps for communication round t from 1 to T:

- **Local training.** The server selects a random set C_t of clients as participants in this round. Each participant $k \in C_t$ then downloads from the server $\mathbf{w}_{t-1}$, the ending global model in round $t-1$. Then, client k performs local updates for local iteration i from 1 to m:

$$\mathbf{w}_{t,i}^k \leftarrow \mathbf{w}_{t,i-1}^k - \eta \nabla_w \mathcal{L}\left(\mathbf{w}_{t,i-1}^k, \mathcal{D}^k\right), \tag{2}$$

  with the starting local model $\mathbf{w}_{t,0}^k$ initialized as $\mathbf{w}_{t-1}$. In addition, η denotes the learning rate, and m denotes the number of local iterations.
- **Model aggregation.** Participants in round t upload their updated local models. The server aggregates the local models to a new global model $\mathbf{w}_t$ by taking a weighted average

$$\mathbf{w}_t \leftarrow \sum_{k \in C_t} \frac{n_k}{N(C_t)} \mathbf{w}_{t,m}^k, \tag{3}$$

  where the weight of client k is the size of k's local training set, namely, n_k.

2.2 Fed-Influence

To express the client-level influence clearly, we introduce a new notation $\mathbf{w}_t\left(C' \rightarrow C' \backslash \{c\}\right)$, where $C' \in \{C_1, C_2, C_3, \ldots, C_T\} \cup \{C\}$, which represents the aggregate model in round t with a client set C' replaced with $C' \backslash \{c\}$. For example, $\mathbf{w}_{10}\left(C_5 \rightarrow C_5 \backslash \{c\}\right)$ represents the ending model we get in round 10 if we remove c in round 5. One special case is that when $C' = C_t$,

$$\mathbf{w}_t\left(C_t \rightarrow C_t \backslash \{c\}\right) = \sum_{k \in C_t \backslash \{c\}} \frac{n_k \mathbf{w}_{t,m}^k}{N\left(C_t \backslash \{c\}\right)}. \tag{4}$$

Another special case is $\mathbf{w}_t\left(C \rightarrow C \backslash \{c\}\right)$, where we permanently remove c, i.e., replace C_t with $C_t \backslash \{c\}$ for all $t \in [T]$.

To quantify the influence of individual clients, we give the following definition of *Fed-Influence on Parameter (FIP).*

Definition 1. *We refer to the change in parameter due to removing a client c from C as Fed-Influence on Parameter (FIP) of client c, denoted by $\epsilon_t^{-c,*}$:*

$$\epsilon_t^{-c,*} \overset{\text{def}}{=} \mathbf{w}_t\left(C \rightarrow C \backslash \{c\}\right) - \mathbf{w}_t. \tag{5}$$

Based on FIP, we can trivially extend the notion to measure the influence on model performance. There are many metrics to evaluate a model's performance, such as accuracy, cross-entropy loss, precision, recall, mean squared error (MSE), etc. We can easily get the influence on any of these metrics once FIP is obtained. Supposing $\mathcal{F}$ is the function for a certain metric over a test set $\mathcal{D}_{test}$, we can compute

$$\mathcal{F}\left(\mathbf{w}_t + \epsilon_t^{-c,*}, \mathcal{D}_{test}\right) - \mathcal{F}\left(\mathbf{w}_t, \mathcal{D}_{test}\right) \tag{6}$$

as the Fed-Influence in terms of the metric. In Sections 5 and 6, we will focus on two most widely used metrics, loss and accuracy, as well as the corresponding two kinds of influence, called Fed-Influence on Loss (FIL) and Fed-Influence on Accuracy (FIA).

The exact FIP of a client can only be obtained by conducting leave-one-out test: retrain the model by removing the client and compare the retrained model with the model trained on the full client set. However, it is prohibitively inefficient to rerun the whole federated learning process, especially when we intend to measure the influence of each client, implying the number of rerunning being the number of all the clients.

3 BASIC ESTIMATOR

We now derive an estimator of $\epsilon_t^{-c,*}$ to avoid retraining. We start by rewriting the expression of $\epsilon_t^{-c,*}$ as follows

$$\begin{aligned}\epsilon_t^{-c,*} &= \mathbf{w}_t\,(C \to C\backslash\{c\}) - \mathbf{w}_t\,(C_t \to C_t\backslash\{c\}) + \mathbf{w}_t\,(C_t \to C_t\backslash\{c\}) - \mathbf{w}_t \\ &= \underbrace{\sum_{k\in C_t\backslash\{c\}} \frac{n_k}{N(C_t\backslash\{c\})} \underbrace{\left(\mathbf{w}_{t,m}^k\,(C \to C\backslash\{c\}) - \mathbf{w}_{t,m}^k\right)}_{\text{local sequential influence}}}_{\text{sequential influence}} + \underbrace{\mathbf{w}_t\,(C_t \to C_t\backslash\{c\}) - \mathbf{w}_t}_{\text{combinatorial influence}}. \end{aligned} \tag{7}$$

Equation 7 shows that $\epsilon_t^{-c,*}$ comprises three parts: (1) *Local sequential influence*: the influence that removing c from C has on the local model of any other client k ($k \neq c$) who has participated in round t. We regard it as "sequential" because it results from $\epsilon_{t-1}^{-c,*}$, the influence in the previous round; (2) *Sequential influence*: the weighted average of local sequential influence; and (3) *Combinatorial influence*: the influence of removing c merely from C_t. It is "combinatorial" because it is independent of $\epsilon_{t-1}^{-c,*}$.

We next dissect how to compute local sequential influence and combinatorial influence. First, the combinatorial one can be easily obtained using Equation 4 and Equation 3. Second regards the local sequential influence. Given that the cause of the difference between $\mathbf{w}_{t,m}^k\,(C \to C\backslash\{c\})$ and $\mathbf{w}_{t,m}^k$ is that they are locally updated from different initial models, $\mathbf{w}_{t-1}\,(C \to C\backslash\{c\})$ and $\mathbf{w}_{t-1}$, respectively, we estimate the term by applying first-order Taylor approximation and the chain rule:

$$\mathbf{w}_{t,m}^k\,(C \to C\backslash\{c\}) - \mathbf{w}_{t,m}^k \approx \frac{\partial \mathbf{w}_{t,m}^k}{\partial \mathbf{w}_{t,m-1}^k} \frac{\partial \mathbf{w}_{t,m-1}^k}{\partial \mathbf{w}_{t,m-2}^k} \cdots \frac{\partial \mathbf{w}_{t,1}^k}{\partial \mathbf{w}_{t,0}^k} \Delta \mathbf{w}_{t,0}^k, \tag{8}$$

where $\Delta \mathbf{w}_{t,0}^k = \mathbf{w}_{t,0}^k\,(C \to C\backslash\{c\}) - \mathbf{w}_{t,0}^k = \epsilon_{t-1}^{-c,*}$. According to the update rule in Equation 2, with the assumption that $\mathcal{L}(\mathbf{w}, \mathcal{D}^k)$ is twice differentiable, we obtain

$$\frac{\partial \mathbf{w}_{t,i}^k}{\partial \mathbf{w}_{t,i-1}^k} = \mathbf{I} - \eta \mathbf{H}_{t,i-1}^k, \tag{9}$$

where $\mathbf{H}_{t,i}^k \overset{\text{def}}{=} \nabla_w^2 \mathcal{L}(\mathbf{w}_{t,i}^k, \mathcal{D}^k)$. By combining Equations 7, 8 and 9, we get an estimator of $\epsilon_t^{-c,*}$, denoted by ϵ_t^{-c}

$$\epsilon_t^{-c} \overset{\text{def}}{=} \mathbf{M}_t^{-c} \epsilon_{t-1}^{-c} + \mathbf{w}_t\,(C_t \to C_t\backslash\{c\}) - \mathbf{w}_t, \tag{10}$$

where

$$\mathbf{M}_t^{-c} \overset{\text{def}}{=} \sum_{k\in C_t\backslash\{c\}} \frac{n_k}{N(C_t\backslash\{c\})} \prod_{i=0}^{m-1} (\mathbf{I} - \eta \mathbf{H}_{t,i}^k). \tag{11}$$

By recalling that the initial model $\mathbf{w}_0$ is randomly initialized by the server, we have $\epsilon_0^{-c} = \mathbf{0}$. Then the estimator ϵ_t^{-c} can be computed iteratively using Equation 10.

We finally take a close look at the relationship between t and the estimation error. We give a uniform bound on the error in both convex and non-convex cases under the following assumptions.

Assumption 1. *There exists λ and Λ such that $\lambda I \preccurlyeq \nabla^2 \mathcal{L} \preccurlyeq \Lambda I$.*

Assumption 2. *The norm of $\epsilon_t^{-c,*}$ is bounded by C, for $t \in [T]$ and $c \in C$.*

Note that in Assumption 1, there is no constraint on the values of λ and Λ except $\lambda \leq \Lambda$, which means the loss function is not necessarily convex. Next we give Theorem 1, the proof of which is deferred to Appendix A.

Theorem 1. *With Assumptions 1 and 2, the error of the estimator is bounded by*

$$\delta_t^{-c} \overset{\text{def}}{=} \|\epsilon_t^{-c,*} - \epsilon_t^{-c}\| \leq \frac{1-\gamma^t}{1-\gamma} o(C), \quad \forall t > 0, \tag{12}$$

where o is the little-o notation, and $\gamma = \alpha^m$, $\alpha = \max\{|1-\eta\lambda|, |1-\eta\Lambda|\}$.

From Equation 12, we can find that the bound is in the format of the sum of geometric series. An intuitive explanation is that each time we use Equation 10, the error in the previous round is scaled by $\mathbf{M}_t^{-c}$ and added to a newly introduced error in this round, just like summing a geometric series. In addition, there are three different cases depending on the relationship between γ and 1:

- **Case 1** ($\gamma < 1$): In this case, $\lambda > 0$ and $0 < \eta < \frac{2}{\Lambda}$, where the loss function is strongly-convex and the learning rate is small enough. This is the most ideal case, where the bound can be further scaled to $\frac{1}{1-\gamma} o(C)$, which is independent of t.
- **Case 2** ($\gamma = 1$): In this case, either $\lambda = 0$ and $\eta \leq \frac{2}{\Lambda}$ or $\lambda \geq 0$ and $\eta = \frac{2}{\Lambda}$. The former situation is more common, with a convex loss function and an appropriate learning rate. Then the error is $o(C)t$, linear with t.
- **Case 3** ($\gamma > 1$): In this case, $\lambda < 0$ or $\eta > \frac{2}{\Lambda}$, where either the loss function is non-convex or the learning rate is too large. Then the error bound is exponential with t, making estimation ineffective.

4 IMPROVING ROBUSTNESS AND EFFICIENCY

In this section, we make some further modifications to the basic estimator from the two aspects: one is to improve the robustness of the method in the non-convex case, and the other is to cut down the high cost brought by computing the Hessian matrix. We also give an illustration of how measurement of influence works in the framework of FedAvg. Algorithm 1 shows the framework of FedAvg in which line 9 is the step added by us to compute FIP using Algorithm 2, where techniques introduced in Section 4.1 and 4.2 are applied.

4.1 Layer-Wise Examination and Truncation (LWET)

The analysis in Section 3 reveals that the basic estimator can have a large error when the loss function is non-convex. Non-convex case is quite common in federated learning for deep learning tasks [6, 22]. We thus propose layer-wise examination and truncation (LWET for short), the details of which and the intuitions are shown as follows.

4.1.1 Truncation. Our primary goal is to avoid an exponential error in Case 3, which results from the estimation of sequential influence. We consider a counterpart, where the sequential influence is completely omitted, i.e., $\epsilon_t^{-c} = \mathbf{w}_t\,(C_t \rightarrow C_t \backslash \{c\}) - \mathbf{w}_t$ (we call it the *truncated estimator* for simplicity), and find that the error becomes independent of t, as shown in the following theorem.

Theorem 2. *With the sequential influence omitted, we get the bound of error as follows:*

$$\delta_t^{-c} \leq \gamma C + o(C).$$

4.1.2 Layer-wise Operation. However, after truncation, a new problem arises: the truncated estimator will always be $\mathbf{0}$ at round t as long as c is not one of the participants. For example, in a setting where 5 clients are selected each round with 100 clients in total, there will be 95 clients with FIP being $\mathbf{0}$ each round, which does not make sense. To ensure accuracy while retaining as much

Algorithm 1 Estimate Fed-Influence on Parameter

```
    Server:
    Output: The final global model $\mathbf{w}_T$; estimated FIP for each client at each round $\epsilon_t^{-c}, \forall c \in C, \forall t \in [T]$
 1: Initialize $\mathbf{w}_0$ with a random model, $\epsilon_0^{-c} = \mathbf{0}$ for $c \in C$, flag variables $e_j = 1$ for $j$ from 1 to number of layers
 2: for synchronization round $t$ from 1 to $T$ do
 3:     $C_t \leftarrow$ random set of clients
 4:     for each client $k \in C_t$ in parallel do
 5:         $\mathbf{w}_{t,m}^k, \mathbf{G}_t^k \leftarrow$ ClientUpdate($\mathbf{w}_{t-1}$)
 6:     end for
 7:     Update the global model $\mathbf{w}_t$ using Equation 3
 8:     for $c \in C$ do
 9:         Call Algorithm 2 to update $\epsilon_t^{-c}$
10:     end for
11: end for

    ClientUpdate:                                        ▷ Run on client $k$
    Input: $\mathbf{w}_{t-1}$ downloaded from the server
    Output: Local model $\mathbf{w}_{t,m}^k$ and a set $\mathbf{G}_t^k$ with each element being a set of $N_s$ local gradients randomly selected out in one local iteration
12: Initialize $\mathbf{w}_{t,0}^k$=$\mathbf{w}_{t-1}$, $\mathbf{G}_t^k = \emptyset$
13: for each local iteration $i$ from 1 to $m$ do
14:     Update the local model $\mathbf{w}_{t,i}^k$ using Equation 2
15:     $\mathcal{G}_{t,i-1}^k \leftarrow \{\nabla_w \mathcal{L}(\mathbf{w}_{t,i-1}^k, z) | z \in \mathcal{S}_{t,i}^k\}$, where $\mathcal{S}_{t,i}^k$ is a subset of $N_s$ samples randomly selected from $\mathcal{D}^k$
16:     $\mathbf{G}_t^k \leftarrow \mathbf{G}_t^k \cup \{\mathcal{G}_{t,i-1}^k\}$
17: end for
```

information as possible, it is necessary to find a happy medium between the basic estimator and the truncated estimator, which instead partially omits the sequential influence. To achieve this, we first introduce layer-wise operation. We calculate only parts of the Hessian matrix, with the interaction between different layers ignored. We take a convolutional neural network (CNN) for example. Supposing that the parameter $\mathbf{w}$ is composed of $\mathbf{w}_{(j)}, j = 1, 2, \ldots, 8$, corresponding to conv-layer 1, bias of conv-layer 1, conv-layer 2, bias of conv-layer 2, dense-layer 1, bias of dense-layer 1, dense-layer 2, bias of dense-layer 2, respectively, i.e., $\mathbf{w} = [\mathbf{w}_{(1)}^T, \mathbf{w}_{(2)}^T, \mathbf{w}_{(3)}^T, \mathbf{w}_{(4)}^T, \mathbf{w}_{(5)}^T, \mathbf{w}_{(6)}^T, \mathbf{w}_{(7)}^T, \mathbf{w}_{(8)}^T]^T$, for each layer j, we see $\mathcal{L}(\mathbf{w})$ as function of $\mathbf{w}_{(j)}$ denoted by $\mathcal{L}_{(j)}(\mathbf{w}_{(j)})$. Rather than computing the entire Hessian matrix $\mathbf{H} = \nabla_w^2 \mathcal{L}(\mathbf{w})$, we only calculate $\mathbf{H}_{(j)} = \nabla_{w_{(j)}}^2 \mathcal{L}_{(j)}(\mathbf{w}_{(j)})$ for each j, some smaller matrices on the diagonal of $\mathbf{H}$. Then the estimator in each layer j is updated independently:

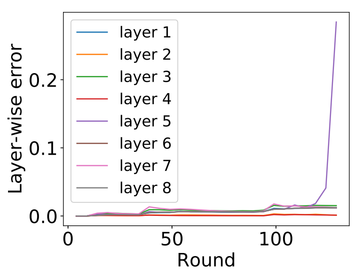
$$\epsilon_{t,(j)}^{-c} \leftarrow \mathbf{M}_{t,(j)}^{-c} \epsilon_{t-1,(j)}^{-c} + \mathbf{w}_{t,(j)}\,(C_t \rightarrow C_t \backslash \{c\}) - \mathbf{w}_{t,(j)},$$

where $\mathbf{M}_{t,(j)}^{-c} = \sum_{k \in C_t \backslash \{c\}} \frac{n_k}{N(C_t \backslash \{c\})} \prod_{i=0}^{m-1} (\mathbf{I} - \eta \mathbf{H}_{t,i,(j)}^k)$. We can examine separately the property of the loss function $\mathcal{L}_{(j)}(\mathbf{w}_{(j)})$ in each layer j, and use a layer-wise truncated estimator shown below only for layers in Case 3:

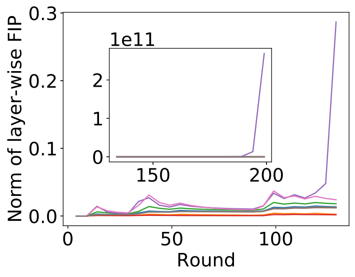
$$\epsilon_{t,(j)}^{-c} \leftarrow \mathbf{w}_{t,(j)}\,(C_t \rightarrow C_t \backslash \{c\}) - \mathbf{w}_{t,(j)}. \tag{13}$$

Algorithm 2 Update FIP

```
    Input: Estimated FIP in the previous round $\epsilon_{t-1}^{-c}$; local models $\mathbf{w}_{t,m}^{k}, \forall k \in C_t$; set of gradient sets $\mathbf{G}_t^k, \forall k \in C_t$; flag variables $e_j$ which indicates whether layer $j$ passes the examination, $j = 1, 2 \ldots,$ number of layers
    Output: FIP in this round $\epsilon_t^{-c}$; flag variables $e_j$
 1: Initialize sequential influence $\sigma_{(j)} = \mathbf{0}$ for $j = 1, 2 \ldots,$ number of layers
 2: for $j$ from 1 to number of layers do
 3:     for $k \in C_t \backslash \{c\}$ do
 4:         Call Algorithm 3 to compute layer-wise local sequential influence $\sigma_{(j)}^{k}$
 5:         $\sigma_{(j)} \leftarrow \frac{n_k}{N(C_t \backslash \{c\})} \sigma_{(j)}^{k} + \sigma_{(j)}$
 6:     end for
 7:     if $e_j$=1 then                                                    ▷ LWET
 8:         if $\|\sigma_{(j)}\| > \|\epsilon_{t-1,(j)}^{-c}\|$ then
 9:             $\epsilon_{t,(j)}^{-c} \leftarrow \mathbf{w}_{t,(j)} (C_t \to C_t \backslash \{c\}) - \mathbf{w}_{t,(j)}$
10:             $e_j \leftarrow 0$
11:         else
12:             $\epsilon_{t,(j)}^{-c} \leftarrow \sigma_{(j)} + \mathbf{w}_{t,(j)} (C_t \to C_t \backslash \{c\}) - \mathbf{w}_{t,(j)}$
13:         end if
14:     else
15:         $\epsilon_{t,(j)}^{-c} \leftarrow \mathbf{w}_{t,(j)} (C_t \to C_t \backslash \{c\}) - \mathbf{w}_{t,(j)}$
16:     end if
17: end for
```



(a) Error



(b) Norm of FIP

Fig. 2. Exponential error and norm. The result of an experiment conducted on setting 2 (see Table 2) where the model is a CNN without activation function. (a) The error and (b) the norm of estimated FIP both have a tendency to increase exponentially on layer 4, the first fully-connected layer. Inset: the norm of estimated FIP from round 135 to 200.

Then we put together layer-wise FIPs to get the complete FIP, i.e., $\epsilon_t^{-c} = [(\epsilon_{t,(1)}^{-c})^T, (\epsilon_{t,(2)}^{-c})^T \ldots]^T$. Because the convexity and continuity of $\mathcal{L}_{(j)}(\mathbf{w}_{(j)})$ vary among different layers, there are layers in Case 1 or Case 2 which still remain the sequential influence. Therefore the complete FIP is non-zero and contains much information even when $c \notin C_t$. In an experiment conducted on a toy model (CNN 2 in Table 2), we found the exponential error only exists in the first fully-connected layer,

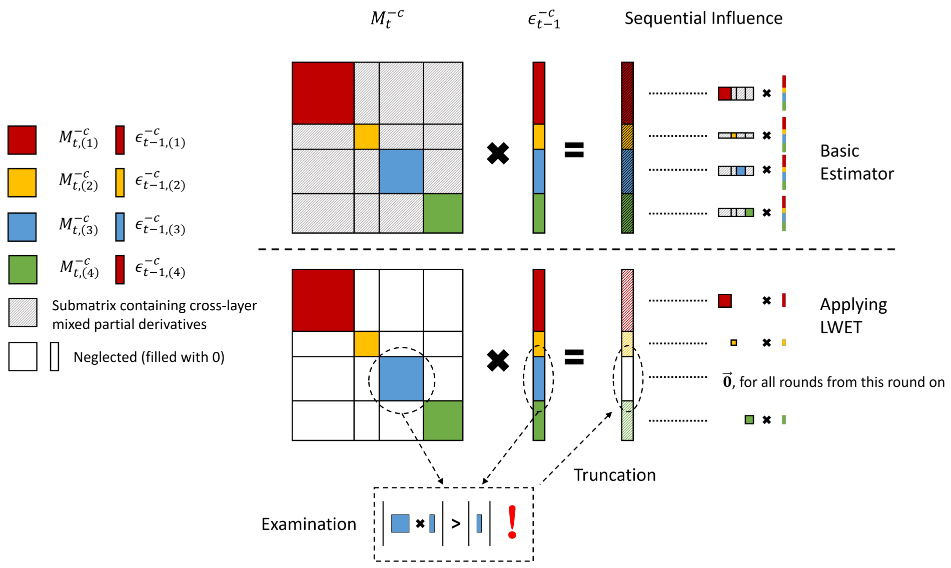

M_t^{-c}
ϵ_{t-1}^{-c}
Sequential Influence
$M_{t,(1)}^{-c}$
$\epsilon_{t-1,(1)}^{-c}$
$M_{t,(2)}^{-c}$
$\epsilon_{t-1,(2)}^{-c}$
$M_{t,(3)}^{-c}$
$\epsilon_{t-1,(3)}^{-c}$
$M_{t,(4)}^{-c}$
$\epsilon_{t-1,(4)}^{-c}$
Submatrix containing cross-layer mixed partial derivatives
Neglected (filled with 0)
Basic Estimator
Applying LWET
$\vec{0}$, for all rounds from this round on
Truncation
Examination

Fig. 3. Illustration of LWET. Here we take a model with four layers of parameters as an example. We give a comparison between the computation of sequential influence in the basic estimator and that in the estimator with LWET applied. In the basic estimator, we simply multiply matrix $\mathbf{M}_t^{-c}$ with vector ϵ_{t-1}^{-c}. By contrast, with LWET, firstly, layer-wise operation itself neglects those submatrices of $\mathbf{M}_t^{-c}$ which contain cross-layer mixed partial derivatives (gray blocks in the figure), and remains only those square submatrices on the diagonal. Then, among the remained submatrices, $\mathbf{M}_{t,(3)}^{-c}$ is further neglected because in this example $\|\mathbf{M}_{t,(3)}^{-c}\epsilon_{t-1,(3)}^{-c}\| > \|\epsilon_{t-1,(3)}^{-c}\|$. More importantly, $\mathbf{M}_{t+1,(3)}^{-c}$, $\mathbf{M}_{t+2,(3)}^{-c}$, ... will also be neglected.

which is the only layer where we need to apply a layer-wise truncated estimator, as shown in Fig. 2(a).

4.1.3 Examination. The next problem is how we can examine the property of loss function in each layer? We propose a mechanism which does not require any *prior knowledge*: in particular, we compare $\|\mathbf{M}_{t,(j)}^{-c}\epsilon_{t-1,(j)}^{-c}\|$ with $\|\epsilon_{t-1,(j)}^{-c}\|$. If $\|\mathbf{M}_{t,(j)}^{-c}\epsilon_{t-1,(j)}^{-c}\|$ is larger at a certain round r, then, in all of the following rounds, i.e., for all $t \geq r$, a layer-wise truncated estimator in Equation 13 will be used in layer j.

This mechanism is aimed at examining a sufficient (but not necessary) condition for $\gamma > 1$. Please refer to Lemma 3 in Appendix A for the reason. In addition, this examination is also avoids the overflow of the estimator itself. The upper bound of ϵ_{t-1}^{-c} is also exponential with t in Case 3, which means there is always a risk for it to overflow as t increases. This phenomenon can be observed in the experiment with the aforementioned toy model. As shown in Fig. 2(b), the norm of estimated FIP on the first fully-connected layer increases sharply around the 125-th round, and then achieves an order of 10^{11} at round 200, indicating the failure of the algorithm.

Combining the three strategies, we get LWET (line 7 to line 17 in Algorithm 2) and give Fig. 3 for easy illustration. Because real-world applications rarely satisfy the Identically and Independently Distributed (IID) assumption and are likely to be non-iid in many ways [16, 17, 23], where clients vary in the data distribution and the property of the local loss function, a more fine-grained version of LWET is recommended in these cases. We can examine the relationship between

Algorithm 3 Approximate Local Sequential Influence

```
    Input: FIP in the previous round $\epsilon_{t-1}^{-c}$; set of gradient sets $\mathbf{G}_t^k$; layer index $j$
    Output: Layer-wise local sequential influence $\sigma_{(j)}^k$
1: $\sigma_{(j)}^k \leftarrow \epsilon_{t-1,(j)}^{-c}$
2: for $\mathcal{G}$ in $\mathbf{G}_t^k$ do
3:     $\mathbf{u} \leftarrow \mathbf{0}$
4:     for $g \in \mathcal{G}$ do
5:         $\beta \leftarrow g_{(j)}^T \sigma_{(j)}^k$
6:         $\mathbf{u}_{(j)} \leftarrow \mathbf{u}_{(j)} + \beta g_{(j)}$
7:     end for
8:     $\sigma_{(j)}^k \leftarrow \sigma_{(j)}^k - \frac{\eta}{N_s}\mathbf{u}_{(j)}$
9: end for
```

$\| \left(\prod_{i=0}^{m-1}(\mathbf{I} - \eta \mathbf{H}_{t,i,(j)}^k)\right) \epsilon_{t,(j)}^{-c}\|$ and $\|\epsilon_{t,(j)}^{-c}\|$. If the former is larger at one round, then we drop the local sequential influence on k from that round on. See this algorithm in Appendix B.

4.2 Low-Cost Hessian Approximation

Computing the Hessian matrix is expensive, requiring $O(p^2)$ operations, where p is the size of the model, and it is not even the most expensive part. The operations where Hessian matrices are involved, matrix-vector multiplication and matrix-matrix multiplication, bring an $O(p^3)$ and $O(p^4)$ complexity, respectively. All of these make the naive implementation prohibitive for large models. Yet, by exploiting the property of the Fisher information and carefully arranging the order of calculation, we design an algorithm with linear computation and communication cost.

4.2.1 Fisher Information. Fisher information is the expectation of the outer product of gradients. Because the cross entropy loss is a negative log-likelihood, the outer product of gradient is equal to the Hessian in expectation, i.e., [5, 18],

$$\mathop{\mathbb{E}}_{z \in \mathcal{D}'}\left[\nabla_w \mathcal{L}(\mathbf{w}, z) \nabla_w \mathcal{L}(\mathbf{w}, z)^T\right] = \mathop{\mathbb{E}}_{z \in \mathcal{D}'}\left[\nabla_w^2 \mathcal{L}(\mathbf{w}, z)\right]. \tag{14}$$

Leveraging the fact that clients holds their own datasets, we let each client randomly select a given number, denoted by N_s, of gradients and use the empirical expectation of the outer product as an approximation to the Hessian:

$$\mathbf{H}_{t,i,(j)}^k \approx \frac{1}{N_s} \sum_{z \in \mathcal{S}_{t,i}^k} g(\mathbf{w}_{t,i,(j)}^k, z) g(\mathbf{w}_{t,i,(j)}^k, z)^T \overset{\text{def}}{=} \tilde{\mathbf{H}}_{t,i,(j)}^k, \tag{15}$$

where $\mathcal{S}_{t,i}^k$ is the set of N_s samples randomly selected by client k at the i-th local iteration in round t, and $g(\mathbf{w}_{t,i,(j)}^k, z) = \nabla_{w_{(j)}} \mathcal{L}_{(j)}(\mathbf{w}_{t,i,(j)}^k, z)$.

4.2.2 Recursive Computation. Simply introducing Fisher information cannot help cut down the cost, because the outer product is $O(p^2)$, and the $O(p^3)$ matrix-vector or $O(p^4)$ matrix-matrix multiplication still exists. However, we can actually circumvent these needless calculations. With

Table 1. Time complexity for the server, time complexity for each client, and communication complexity for each client. n is the size of the local dataset and $K = |C|$. In a naive implementation, operations where the Hessian matrix is involved, can be taken either on clients or the server, corresponding to naive implementation 1 and 2, respectively. For naive implementation 1, each client k has to compute $\prod_{i=0}^{m-1}(\mathbf{I} - \eta \mathbf{H}_{t,i}^k)$ locally, which requires matrix-matrix multiplications, and upload the result to the server. For naive implementation 2, each client k upload $\mathbf{H}_{t,i}^k$ for all i, and then the server compute $\mathbf{M}_t^{-c}\epsilon_{t-1}^{-c}$, which can be implemented with only matrix-vector multiplications. We also give the cost of a pure learning process.

	Server	Client	Communication
Naive implementation 1	K^2p^2	$m(np^2 + p^4)$	p^2
Naive implementation 2	K^2mp^3	mnp^2	mp^2
Our method	K^2mN_sp	mnp	mN_sp
Pure learning	Kp	mnp	p

$\mathbf{H}_{t,i,(j)}^k$ approximated by $\tilde{\mathbf{H}}_{t,i,(j)}^k$, the estimator of local sequential influence is

$$\left(\prod_{i=0}^{m-1}\left(\mathbf{I} - \eta \frac{1}{N_s}\sum_{z\in\mathcal{S}_{t,i}^k} g(\mathbf{w}_{t,i,(j)}^k, z) g(\mathbf{w}_{t,i,(j)}^k, z)^T\right)\right)\epsilon_{t-1,(j)}^{-c}. \tag{16}$$

Instead of first calculating the cumprod on the left and then multiplying it by $\epsilon_{t-1,(j)}^{-c}$, the linear-cost method is based on a recursive computation: we initialize a vector $\sigma_{(j)}^k$ by $\sigma_{(j)}^k \leftarrow \epsilon_{t-1,(j)}^{-c}$, and then we update $\sigma_{(j)}^k$ using Equation 17 repeatedly for i from 0 to $m-1$:

$$\sigma_{(j)}^k \leftarrow \sigma_{(j)}^k - \frac{\eta}{N_s}\sum_{z\in\mathcal{S}_{t,i}^k} g(\mathbf{w}_{t,i,(j)}^k, z)\left(g(\mathbf{w}_{t,i,(j)}^k, z)^T \sigma_{(j)}^k\right). \tag{17}$$

$\sigma_{(j)}^k$ produced in the final iteration is the value of Equation 16. We give the corresponding algorithm in Algorithm 3. It worth noting that this method requires the computation to take place on the server, because only the server has $\epsilon_{t-1,(j)}^{-c}$, and clients need to do nothing other than randomly select and upload a certain number of local gradients (line 15 and 16 in Algorithm 1). We show the efficiency of our method by comparing it with naive implementations in Table 1.

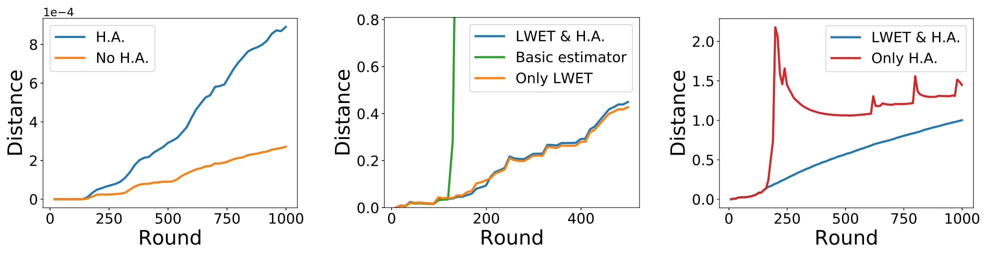
5 FUNDAMENTAL EXPERIMENTS

In this section, we mainly demonstrate two aspects of our method: (1) LWET plays a vital role; and (2) Hessian approximation causes only a slight drop in the accuracy. Experiments are conducted on 64bit Ubuntu 18.04 LTS with four Intel i9-9900K CPUs and two NVIDIA RTX-2080TI GPUs, 200GB storage. We take “Leaf" [1], a benchmarking framework for federated learning based on tensorflow. We evaluated our method on three settings, as shown in Table 2. We used the softmax function at the output layer and adopted the cross entropy as the loss function. In setting 1, the loss function is convex but not strongly convex, and therefore it is in Case 2 ($\gamma = 1$). In setting 2, although the toy model has no activation function, which makes it equivalent to a single-layer perceptron with convex loss function, results show that it is still in Case 3 ($\gamma > 1$) because the learning rate is too large. And in setting 3, the loss function is non-convex and is therefore in Case 3, too.

We use four different methods to obtain ϵ_t^{-c}: (1) the basic estimator, (2) the estimator with only LWET, (3) the estimator with only Hessian approximation, and (4) the estimator with both LWET and Hessian approximation. We do not demonstrate results of all methods in each setting. In

Table 2. Configuration of the three different settings. The two datasets are described in Caldas et al. [1]. The logistic regression model is the original one in "Leaf". We made a little adjustment to the original CNN to create models with certain properties and scales. Detailed descriptions of data distributions and CNN structures are given in Appendices C and D.

	Model	Dateset	Distribution	η	$\lvert C\rvert$	$\lvert C_t\rvert$	m	T	N_s
Setting 1	LogReg	Synthetic	Non-IID, Unbalance	0.003	1000	10	5	1000	50
Setting 2	CNN 1	FEMNIST	IID, Balance	0.03	50	5	2	500	50
Setting 3	CNN 2	FEMNIST	Non-IID, Unbalance	0.02	100	10	2	2000	50



(a) Setting 1 (b) Setting 2 (c) Setting 3

Fig. 4. The distance between exact FIP and estimated FIP in the three settings.

setting 1, we had tested all of the four methods, but found that the result produced with LWET is completely the same with that produced without LWET, which further validates that setting 1 is in Case 2. Therefore we only show two different results, the result based on exact Hessian and that based on approximated Hessian. In setting 2, we do not demonstrate the method with only Hessian approximation, because the basic estimator has already caused a terrible error, and the introduction of Hessian approximation will, no doubt, create an even larger error. In setting 3, we can only test the two methods that contain a Hessian approximation because the large model size and the limited resources on our device do not allow us to compute the exact Hessian.

5.1 Influence on Parameter

We examine the error of the proposed method $\|\epsilon_t^{-c,*} - \epsilon_t^{-c}\|$, which is the distance between exact FIP and estimated FIP under L_2 norm. The exact FIP is obtained by conducting leave-one-out tests. We track the error of one randomly selected client's FIP and show the result in Fig. 4. We can see from Fig. 4(b) the necessity of LWET in setting 2: without LWET there will be a sharp increase in the error at about the 120-th round (the error is caused by the first fully-connected layer as mentioned earlier in Fig. 2). In setting 3, as shown in Fig. 4(c), although it seems that without LWET the error won't be extremely large like that in setting 2, results given in the next subsection indicate that we still should take LWET to improve the result which will otherwise run meaningless. As for the effect of Hessian approximation, results in Figures 4(a) and 4(b) show that it only causes a slight increase in the error.

5.2 Influence on Loss

In this section, we give a more intuitive demonstration of the experimental results by mapping the high-dimensional FIP to a scalar, FIL. The exact FIL is obtained from the result of leave-one-out test.

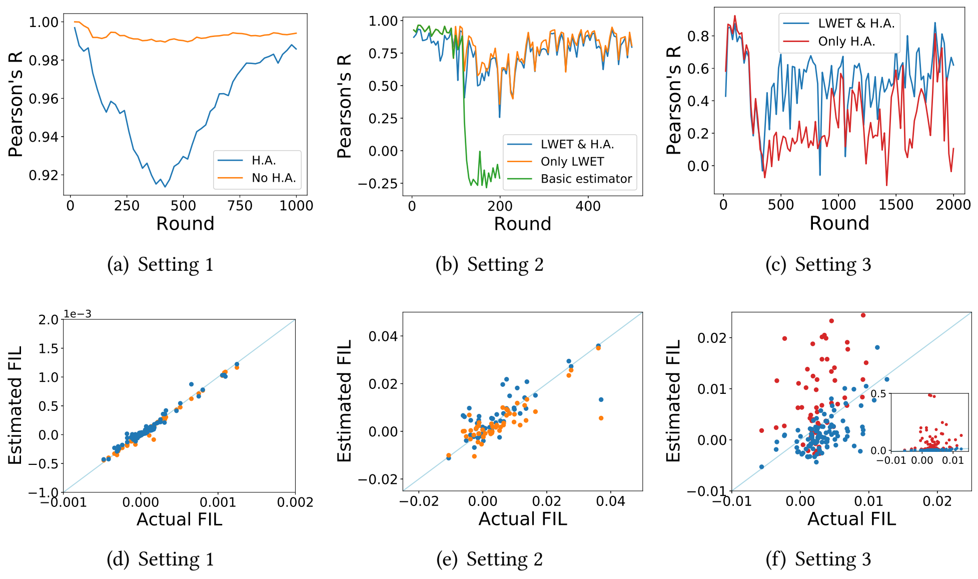


Fig. 5. (a)(b)(c) The change of Pearson coefficient over rounds in setting 1, 2 and 3. (d)(e)(f) Estimated FIL vs. actual FIL in setting 1, 2 and 3. The inset in (f) shows the results with a wider y-axis range. In setting 1, we counted 200 clients that were randomly selected from the 1000 clients. In settings 2 and 3, we counted all the clients, 50 and 100, respectively.

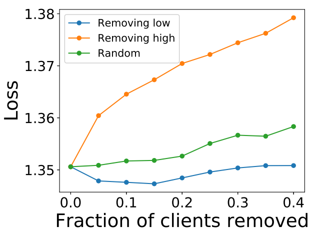
By adding estimated FIP to the original global model, we get a estimation of the model trained with one client removed $\mathbf{w}_t + \epsilon_t^{-c}$. Then we test it on $\mathcal{D}_{test}$ and subtract from the result the loss of the original model to get the estimated FIL, i.e., $\mathcal{L}(\mathbf{w}_t + \epsilon_t^{-c}, \mathcal{D}_{test}) - \mathcal{L}(\mathbf{w}_t, \mathcal{D}_{test})$.

We compare the estimated FIL with the exact FIL and show the results in Fig. 5. The correlation between the estimated and exact FIL is measured by Pearson's correlation coefficient, and we plot its variation with time in Figures 5(a), 5(b), and 5(c). We also visualize the correlation in the last round in Figures 5(d), 5(e), and 5(f). In particular, the proposed method achieves a Pearson correlation coefficient of 0.6200 at the last round in setting 3, the most difficult setting; in setting 1 Pearson correlation coefficient is 0.9857 and in setting 2 it is 0.7957.

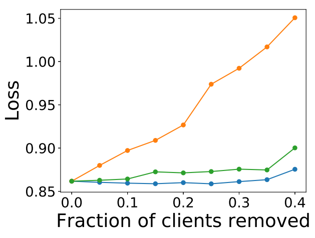
6 EXTENDED EXPERIMENTS

In this section we use FIL and FIA for client valuation and client cleansing, despite that we believe there are more potential applications based on other information given by FIP.

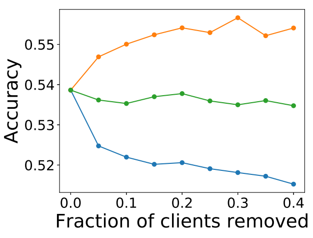
Influence on model performance can be used as a reasonable metric to determine a client's value. We conduct the following experiment: we make an observation at clients' influence at a certain round, remove some of the clients with highest/lowest influence, from the training set, and then continue the learning process. The experiment is repeated with different fractions of clients removed and the performance of the final global model is recorded each time. As can be seen from Fig. 6, removing valuable clients (those with high FIL or low FIA) greatly compromises the model performance, which indicates the importance of these clients. In contrast, removing least valuable clients (those with low FIL or high FIA) improves the model performance. And the curve of randomly removing falls between the other two curves.



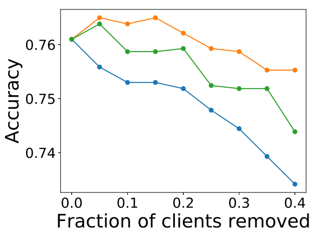
(a) Setting 1: Loss



(b) Setting 2: Loss



(c) Setting 1: Accuracy



(d) Setting 2: Accuracy

Fig. 6. We remove clients from the client set at a certain round x in three different ways: from those with lowest influence, from those with highest influence, randomly. (a)(b) We use FIL and record the corresponding change in loss. (c)(d) We use FIA and record the corresponding change in accuracy. (a)(c) The results in setting 1 with $x = 700$. (b)(d) The results in setting 3 with $x = 1500$.

The results of removing least valuable clients elicits an application, which we call client cleansing. It is a little different from the data cleansing in Hara et al. [9]. Data cleansing is conducted by retraining the model with a subset of data removed. In the setting of federated learning, as we mentioned before, it does not make sense to retrain the model in most real-world scenarios. Therefore, we conduct client cleansing by removing a subset of clients at a certain round and then continue the training. By properly selecting the clients to be removed we can effectively improve the final model. In setting 1, the accuracy is increased from 53.86% to 55.66% by removing 30% of the clients from the client set at round 700. In setting 3, the accuracy is increased from 76.10% to 76.50% by removing 5% of the clients from the client set at round 1500. However, more investigations are required to deicide the best timing to conduct client cleansing. By observing the trends in clients' FIL-based rankings (Appendix E) we make some intuitive conclusions. On one hand, to get a reasonable valuation of clients, we don't have to wait for too long, because clients' influence-based rankings become stable after the training has progressed to a certain stage, especially for those among the highest or lowest, and removing the right clients as early as possible can minimize their negative effects on the model. However, on the other hand, cleansing should not be conducted too early, because the rankings at that time are meaningless and may lead us to make a wrong decision.

7 DISCUSSIONS

The proposed estimator of FIP in Section 3 includes that proposed in Hara et al. [9] as its special case, which is aimed at estimating sample-level influence in SGD (called SGD-Influence in their paper). In federated learning, if each client holds only 1 sample and takes only 1 step of local update, then the mathematical expression of the training process is consistent with that in SGD, in which case clients are equivalent to individual samples, the client set selected each round is equivalent to the mini batch selected each step and the estimator in Equation 10 is also equivalent to the estimator of SGD-Influence.

However, the proposed implementation is completely different from that in SGD. The key difference is that Hara et al. focused on influence on loss, while we focus on influence on parameter. In their implementation, the considerable time and space complexity that result from Hessian matrix $\mathbf{H}_{t,i}^{k}$ can be reduced by directly computing the derivative of $\langle \mu, \nabla_w \mathcal{L}(\mathbf{w}_{t,i}^{k}, \mathcal{D}_k)\rangle$, called LIE (Linear Influence Estimation) in their paper. By setting $\mu = \nabla_w \mathcal{L}(\mathbf{w}_t, \mathcal{D}_{test})$, the inner product amounts to a linearly approximated influence on loss. Although it works well in centralized learning, we show that it is impractical to apply this trick in federated learning:

- First, the derivative of the inner product $\langle \nabla_w \mathcal{L}(\mathbf{w}_t, \mathcal{S}_{test}), \nabla_w \mathcal{L}(\mathbf{w}_{t,i}^{k}, \mathcal{D}_k)\rangle$ requires the local data of client k, thus it can only be computed locally by k during its local training in round t, since the server has no access to private data. However, at round t, k has no idea about $\mathbf{w}_t$, because $\mathbf{w}_t$ is the global model which can only be computed on the server at the end of round t, when all the participating clients in this round have finished local updates and uploaded their local models.
- Second, what about the server sending $\nabla_w \mathcal{L}(\mathbf{w}_t, \mathcal{S}_{test})$ to clients? This idea is, in essence, an extension of Hara et al. [9]'s method from SGD to federated learning. Here we describe it in the context of federated learning. This method contains two phases, the training phase and the inference phase: in the training phase, the client sets C_t and the local models $\mathbf{w}_{t,i}^{k}$ are stored; in the inference phase, the stored information is retraced to compute $\langle \nabla_w \mathcal{L}(\mathbf{w}_t, \mathcal{S}_{test}), \nabla_w \mathcal{L}(\mathbf{w}_{t,i}^{k}, \mathcal{D}_k)\rangle$. However, this does not make sense in federated learning for mainly two reasons: (1) Retracing is much more difficult than training, because in the training phase, we randomly select clients from those already accessible, while when retracing them, we have to re-communicate with given ones in the given order. We recall that clients are not always accessible. For example, there's no guarantee that the client we need has been connected to the Internet, due to the erratic network condition or whatever reason, when we try to communicate with it in the inference phase; and (2) this method requires clients to store the models produced at every local step, i.e., $\mathbf{w}_{t,i}^{k}$ for all t and i. However, a client's device cannot hold this much information due to the limited space.

We can conclude that the conditions in federated learning do not allow us to directly obtain influence on loss without computing influence on parameter. For this reason we focused on measuring the influence on parameter in parallel with the training process, which is more compatible with the framework of federated learning.

8 CONCLUSION

In this paper, we proposed Fed-Influence as a new metric of clients' influence on the global model in federated learning, inspired by the classical statistics notion of influence in centralized learning. The differences between the centralized learning and federated learning create challenges in calculating this new metric. To develop an efficient implementation, we first proposed a basic estimator to calculate the individual client's influence on parameter, then further revised the estimator and established an algorithm with both robustness and linear cost. It is worth noting that the proposed method is accurate without assumption on convexity or data identity, which is validated by the empirical results on different settings. We also demonstrated how Fed-Influence helps evaluate clients and improve the model performance through client cleansing. Our work provides a quantitative insight into the relationship between individual clients and the global model, we envision which further enhancing the accountability and interpretability of federated learning.

REFERENCES

[1] Sebastian Caldas, Peter Wu, Tian Li, Jakub Konecný, H. Brendan McMahan, Virginia Smith, and Ameet Talwalkar. 2018. LEAF: A Benchmark for Federated Settings. *CoRR* (2018).

[2] R. Dennis Cook and Sanford Weisberg. 1980. Characterizations of an empirical influence function for detecting influential cases in regression. *Technometrics* (1980), 495–508.

[3] R Dennis Cook. 1977. Detection of influential observation in linear regression. *Technometrics* (1977), 15–18.

[4] R Dennis Cook and Sanford Weisberg. 1982. *Residuals and influence in regression.*

[5] Jerome Friedman, Trevor Hastie, and Robert Tibshirani. 2001. *The elements of statistical learning.*

[6] Farzin Haddadpour, Mohammad Mahdi Kamani, Mehrdad Mahdavi, and Viveck R. Cadambe. 2019. Local SGD with Periodic Averaging: Tighter Analysis and Adaptive Synchronization. In *NeurIPS*. 11080–11092.

[7] Jenny Hamer, Mehryar Mohri, and Ananda Theertha Suresh. 2020. FedBoost: Communication-Efficient Algorithms for Federated Learning.

[8] Frank R Hampel. 1974. The influence curve and its role in robust estimation. *Journal of the american statistical association* (1974), 383–393.

[9] Satoshi Hara, Atsushi Nitanda, and Takanori Maehara. 2019. Data Cleansing for Models Trained with SGD. In *NeurIPS*. 4215–4224.

[10] Louis A Jaeckel. 1972. *The infinitesimal jackknife.*

[11] Peter Kairouz, H. Brendan McMahan, Brendan Avent, Aurélien Bellet, Mehdi Bennis, Arjun Nitin Bhagoji, Keith Bonawitz, Zachary Charles, Graham Cormode, Rachel Cummings, Rafael G. L. D'Oliveira, Salim El Rouayheb, David Evans, Josh Gardner, Zachary Garrett, Adrià Gascón, Badih Ghazi, Phillip B. Gibbons, Marco Gruteser, Zaïd Harchaoui, Chaoyang He, Lie He, Zhouyuan Huo, Ben Hutchinson, Justin Hsu, Martin Jaggi, Tara Javidi, Gauri Joshi, Mikhail Khodak, Jakub Konecný, Aleksandra Korolova, Farinaz Koushanfar, Sanmi Koyejo, Tancrède Lepoint, Yang Liu, Prateek Mittal, Mehryar Mohri, Richard Nock, Ayfer Özgür, Rasmus Pagh, Mariana Raykova, Hang Qi, Daniel Ramage, Ramesh Raskar, Dawn Song, Weikang Song, Sebastian U. Stich, Ziteng Sun, Ananda Theertha Suresh, Florian Tramèr, Praneeth Vepakomma, Jianyu Wang, Li Xiong, Zheng Xu, Qiang Yang, Felix X. Yu, Han Yu, and Sen Zhao. 2019. Advances and Open Problems in Federated Learning. *CoRR* (2019).

[12] Sai Praneeth Karimireddy, Satyen Kale, Mehryar Mohri, Sashank J. Reddi, Sebastian U. Stich, and Ananda Theertha Suresh. 2019. SCAFFOLD: Stochastic Controlled Averaging for On-Device Federated Learning. *CoRR* (2019).

[13] Rajiv Khanna, Been Kim, Joydeep Ghosh, and Sanmi Koyejo. 2019. Interpreting Black Box Predictions using Fisher Kernels. In *AISTATS*. 3382–3390.

[14] Pang Wei Koh, Kai-Siang Ang, Hubert H. K. Teo, and Percy Liang. 2019. On the Accuracy of Influence Functions for Measuring Group Effects. In *NeurIPS*. 5255–5265.

[15] Pang Wei Koh and Percy Liang. 2017. Understanding Black-box Predictions via Influence Functions. In *Proc. of ICML*. 1885–1894.

[16] Tian Li, Anit Kumar Sahu, Manzil Zaheer, Maziar Sanjabi, Ameet Talwalkar, and Virginia Smith. 2020. Federated Optimization in Heterogeneous Networks. In *Proc. of MLSys*.

[17] Xiang Li, Kaixuan Huang, Wenhao Yang, Shusen Wang, and Zhihua Zhang. 2020. On the Convergence of FedAvg on Non-IID Data. In *ICLR*.

[18] Alexander Ly, Maarten Marsman, Josine Verhagen, Raoul PPP Grasman, and Eric-Jan Wagenmakers. 2017. A tutorial on Fisher information. *Journal of Mathematical Psychology* (2017), 40–55.

[19] Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Agüera y Arcas. 2017. Communication-Efficient Learning of Deep Networks from Decentralized Data. In *Proc. of AISTATS*. 1273–1282.

[20] Daniel Rothchild, Ashwinee Panda, Enayat Ullah, Nikita Ivkin, Ion Stoica, Vladimir Braverman, Joseph Gonzalez, and Raman Arora. 2020. FetchSGD: Communication-Efficient Federated Learning with Sketching. *CoRR* (2020).

[21] Jianyu Wang, Qinghua Liu, Hao Liang, Gauri Joshi, and H. Vincent Poor. 2020. Tackling the Objective Inconsistency Problem in Heterogeneous Federated Optimization. *CoRR* (2020).

[22] Hao Yu, Sen Yang, and Shenghuo Zhu. 2019. Parallel Restarted SGD with Faster Convergence and Less Communication: Demystifying Why Model Averaging Works for Deep Learning. In *AAAI*. 5693–5700.

[23] Yue Zhao, Meng Li, Liangzhen Lai, Naveen Suda, Damon Civin, and Vikas Chandra. 2018. Federated Learning with Non-IID Data. *CoRR* (2018).

A PROOFS

Lemma 1. *For real symmetric matrices A and B, if $-\psi_1\mathbf{I} \preccurlyeq A \preccurlyeq \psi_1\mathbf{I}$ and $-\psi_2\mathbf{I} \preccurlyeq B \preccurlyeq \psi_2\mathbf{I}$, then $-\psi_1\psi_2\mathbf{I} \preccurlyeq AB \preccurlyeq \psi_1\psi_2\mathbf{I}$.*

Proof. Firstly we have that $\psi_1\mathbf{I} - A$, $\psi_2\mathbf{I} + B$, $\psi_1\mathbf{I} + A$ and $\psi_2\mathbf{I} - B$ are all real symmetric positive semidefinite matrices. Then, since the sum and product of two real symmetric positive semidefinite matrices are both real symmetric positive semidefinite, we have

$$\begin{aligned}\psi_1\psi_2\mathbf{I} - AB &= \frac{1}{2}(\psi_1\mathbf{I} - A)(\psi_2\mathbf{I} + B) + \frac{1}{2}(\psi_1\mathbf{I} + A)(\psi_2\mathbf{I} - B) \succcurlyeq O \\ \psi_1\psi_2\mathbf{I} + AB &= \frac{1}{2}(\psi_1\mathbf{I} + A)(\psi_2\mathbf{I} + B) + \frac{1}{2}(\psi_1\mathbf{I} - A)(\psi_2\mathbf{I} - B) \succcurlyeq O,\end{aligned} \tag{18}$$

where O is the zero matrix. Then we conclude $-\psi_1\psi_2\mathbf{I} \preccurlyeq AB \preccurlyeq \psi_1\psi_2\mathbf{I}$. □

Lemma 2. *For real symmetric matrix A, if $-\psi\mathbf{I} \preccurlyeq A \preccurlyeq \psi\mathbf{I}$, then $\|A\mathbf{v}\| \leq \psi\|\mathbf{v}\|, \forall \mathbf{v}$.*

Proof. Applying Lemma 1 we have $AA \preccurlyeq \psi^2\mathbf{I}$, which means $\mathbf{v}^T AA\mathbf{v} \leq \psi^2\mathbf{v}^T\mathbf{v}, \forall\mathbf{v}$. Rewrite it as $\|A\mathbf{v}\|^2 \leq \psi^2\|\mathbf{v}\|^2$, i.e. $\|A\mathbf{v}\| \leq \psi\|\mathbf{v}\|$ □

Lemma 3. *With Assumption 1, $\mathbf{M}_t^{-c}$ has the following property*

$$\|\mathbf{M}_t^{-c}\mathbf{v}\| \leq \gamma\|\mathbf{v}\|, \quad \forall\mathbf{v}, \forall t > 0 \tag{19}$$

where $\gamma = \alpha^m$ with $\alpha = \max\{|1 - \eta\lambda|, |1 - \eta\Lambda|\}$.

Proof. From Assumption 1 we obtain $(1 - \eta\Lambda)\mathbf{I} \preccurlyeq \mathbf{I} - \eta\mathbf{H}_{t,i}^k \preccurlyeq (1 - \eta\lambda)\mathbf{I}$. Further scaling it yields $-\alpha\mathbf{I} \preccurlyeq \mathbf{I} - \eta\mathbf{H}_{t,i}^k \preccurlyeq \alpha\mathbf{I}$. Then we can apply Lemma 1 to obtain $-\gamma\mathbf{I} \preccurlyeq \prod_{i=0}^{m-1}(\mathbf{I} - \eta\mathbf{H}_{t,i}^k) \preccurlyeq \gamma\mathbf{I}$. According to Equation 11, $\mathbf{M}_t^{-c}$ is the weighted average of $\prod_{i=0}^{m-1}(\mathbf{I} - \eta\mathbf{H}_{t,i}^k)$, thus $-\gamma\mathbf{I} \preccurlyeq \mathbf{M}_t^{-c} \preccurlyeq \gamma\mathbf{I}$. Finally, we apply Lemma 2 to conclude $\|\mathbf{M}_t^{-c}\mathbf{v}\| \leq \gamma\|\mathbf{v}\|$. □

Proof of Theorem 1. We start by reviewing the derivation of the estimator. We see $\mathbf{w}_{t,m}^k$ as a function of $\mathbf{w}_{t,0}^k$, i.e. $\mathbf{w}_{t,m}^k = h(\mathbf{w}_{t,0}^k)$, so $\mathbf{w}_{t,m}^k(C \rightarrow C\backslash\{c\}) = h(\mathbf{w}_{t,0}^k + \epsilon_{t-1}^{-c,*})$. Then we recall Equation 8 and 11 that we have taken a first-order Taylor approximation for the local sequential influence, i.e. we use $\frac{\partial h(\mathbf{w}_{t,0}^k)}{\partial \mathbf{w}_{t,0}^k}\epsilon_{t-1}^{-c,*}$ as an estimator of $h(\mathbf{w}_{t,0}^k + \epsilon_{t-1}^{-c,*}) - h(\mathbf{w}_{t,0}^k)$ with an error $o(\|\epsilon_{t-1}^{-c,*}\|)$, and that is why inequality 21 holds. Equality 20 comes from the fact that $\mathbf{M}_t^{-c}$ can be expressed as

$\sum_{k\in C_t\backslash\{c\}} \frac{n_k}{N(C_t\backslash\{c\})}\frac{\partial \mathbf{w}^k_{t,m}}{\partial \mathbf{w}^k_{t,0}}$. Inequality 22 comes from Lemma 3. In inequality 23 we use Assumption 2.

$$
\begin{aligned}
\delta_t^{-c} &= \left\|\epsilon_t^{-c} - \epsilon_t^{-c,*}\right\| \\
&= \left\| \mathbf{M}_t^{-c}\epsilon_{t-1}^{-c} - \sum_{k\in C_t\backslash\{c\}} \frac{n_k}{N(C_t\backslash\{c\})}\left(\mathbf{w}^k_{t,m}\left(C\to C\backslash\{c\}\right) - \mathbf{w}^k_{t,m}\right)\right\| \\
&\le \left\|\mathbf{M}_t^{-c}\epsilon_{t-1}^{-c} - \mathbf{M}_t^{-c}\epsilon_{t-1}^{-c,*}\right\| + \left\|\mathbf{M}_t^{-c}\epsilon_{t-1}^{-c,*} - \sum_{k\in C_t\backslash\{c\}} \frac{n_k}{N(C_t\backslash\{c\})}\left(\mathbf{w}^k_{t,m}\left(C\to C\backslash\{c\}\right) - \mathbf{w}^k_{t,m}\right)\right\| \\
&= \left\|\mathbf{M}_t^{-c}\left(\epsilon_{t-1}^{-c} - \epsilon_{t-1}^{-c,*}\right)\right\| \\
&\quad + \left\| \sum_{k\in C_t\backslash\{c\}} \frac{n_k}{N(C_t\backslash\{c\})}\left(\frac{\partial \mathbf{w}^k_{t,m}}{\partial \mathbf{w}^k_{t,0}}\epsilon_{t-1}^{-c,*} - \left(\mathbf{w}^k_{t,m}\left(C\to C\backslash\{c\}\right) - \mathbf{w}^k_{t,m}\right)\right)\right\| \qquad (20) \\
&\le \left\|\mathbf{M}_t^{-c}\left(\epsilon_{t-1}^{-c} - \epsilon_{t-1}^{-c,*}\right)\right\| + \sum_{k\in C_t\backslash\{c\}} \frac{n_k}{N(C_t\backslash\{c\})}\underbrace{\left\|\frac{\partial \mathbf{w}^k_{t,m}}{\partial \mathbf{w}^k_{t,0}}\epsilon_{t-1}^{-c,*} - \left(\mathbf{w}^k_{t,m}\left(C\to C\backslash\{c\}\right) - \mathbf{w}^k_{t,m}\right)\right\|}_{\text{error of first-order Taylor approximation}} \\
&\le \left\|\mathbf{M}_t^{-c}\left(\epsilon_{t-1}^{-c} - \epsilon_{t-1}^{-c,*}\right)\right\| + o(\|\epsilon_{t-1}^{-c,*}\|) \qquad (21) \\
&\le \gamma\delta_{t-1}^{-c} + o(\|\epsilon_{t-1}^{-c,*}\|) \qquad (22) \\
&\le \gamma\delta_{t-1}^{-c} + o(C) \qquad (23)
\end{aligned}
$$

By recursively applying the scaling, we obtain Theorem 1. □

Proof of Theorem 2. If ϵ_t^{-c} is redefined as $\mathbf{w}_t\left(C_t \to C_t\backslash\{c\}\right) - \mathbf{w}_t$, then the error becomes sequential influence itself. To compare it with the error of the basic estimator, we decompose it in terms of its first-order Taylor approximation. Then we can show the proof in a similar way to the proof of Theorem 1.

$$
\begin{aligned}
\delta_t^{-c} &= \left\| \sum_{k\in C_t\backslash\{c\}} \frac{n_k}{N(C_t\backslash\{c\})} \left(\mathbf{w}_{t,m}^k (C \to C\backslash\{c\}) - \mathbf{w}_{t,m}^k\right) \right\| \\
&= \left\| \sum_{k\in C_t\backslash\{c\}} \frac{n_k}{N(C_t\backslash\{c\})} \left(\frac{\partial \mathbf{w}_{t,m}^k}{\partial \mathbf{w}_{t,0}^k} \epsilon_{t-1}^{-c,*} - \frac{\partial \mathbf{w}_{t,m}^k}{\partial \mathbf{w}_{t,0}^k} \epsilon_{t-1}^{-c,*} + \mathbf{w}_{t,m}^k (C \to C\backslash\{c\}) - \mathbf{w}_{t,m}^k \right) \right\| \\
&\le \left\| \sum_{k\in C_t\backslash\{c\}} \frac{n_k}{N(C_t\backslash\{c\})} \frac{\partial \mathbf{w}_{t,m}^k}{\partial \mathbf{w}_{t,0}^k} \epsilon_{t-1}^{-c,*} \right\| \\
&\quad + \sum_{k\in C_t\backslash\{c\}} \frac{n_k}{N(C_t\backslash\{c\})} \underbrace{\left\| \frac{\partial \mathbf{w}_{t,m}^k}{\partial \mathbf{w}_{t,0}^k} \epsilon_{t-1}^{-c,*} - \left(\mathbf{w}_{t,m}^k (C \to C\backslash\{c\}) - \mathbf{w}_{t,m}^k\right) \right\|}_{\text{error of first-order Taylor approximation}} \qquad (24) \\
&\le \left\| \sum_{k\in C_t\backslash\{c\}} \frac{n_k}{N(C_t\backslash\{c\})} \frac{\partial \mathbf{w}_{t,m}^k}{\partial \mathbf{w}_{t,0}^k} \epsilon_{t-1}^{-c,*} \right\| + o(C) \\
&= \left\| \mathbf{M}_t^{-c} \epsilon_{t-1}^{-c,*} \right\| + o(C) \\
&\le \gamma \| \epsilon_{t-1}^{-c,*} \| + o(C) \\
&\le \gamma C + o(C)
\end{aligned}
$$

□

B LWET FOR NON-IID SETTINGS

In non-convex cases where clients are likely to vary in the value of γ, in order to avoid omitting the sequential influence in a layer just because several clients' unqualified γ, we should take the examination on the local sequential influence. To implement this, Algorithm 2 should be replaced by Algorithm 4, and in line 1 of Algorithm 1, flag variables should be e_j^k instead of e_j. We used this method when conducting experiments in setting 3.

C DATASET DESCRIPTION

As is shown in Table 2, there are three datasets, all of which are produced using "Leaf" [1]. In setting 1, the initial data are generated by the process given in "Leaf" with the default setting: the number of data dimensions set to 60 and number of classes set to 5. The final data split is then produced in the non-iid and unbalanced manner. We show the statistics of its distribution in Fig. 7(a). The FEMNIST dataset in setting 2 has an iid and balanced distribution, with 50 clients each holding 50 training samples. The FEMNIST dataset in setting 3 is non-iid and unbalanced, the statistics of which is given in Fig. 7(b).

D STRUCTURES OF CONVOLUTIONAL NEURAL NETWORKS

We made a little adjustment to the original CNN in "Leaf" to create models with certain properties and scales.

- CNN1 is a toy model without activation function: [Input, $x \in \mathbb{R}^{28\times28\times1}$]→[Conv2D, size=5 × 5 × 4, padding="same"]→[MaxPool2D, size=2 × 2, strides=2]→[Conv2D, size=5 × 5 × 4,

Algorithm 4 Update FIP

```
Input: FIP in the previous round $\epsilon_{t-1}^{-c}$; local models $\mathbf{w}_{t,m}^k, \forall k \in C_t$; set of gradient sets $\mathbf{G}_t^k, \forall k \in C_t$; flag variables $e_j^k$ which indicates whether client $k$'s local layer-$j$ loss function passes the examination, $j = 1, 2 \ldots, number\ of\ layers, \forall k \in C_t$
Output: FIP in this round $\epsilon_t^{-c}$; flag variables $e_j^k$
 1: Initialize sequential influence $\sigma_{(j)} = \mathbf{0}$ for $j = 1, 2 \ldots, number\ of\ layers$
 2: for $j$ from 1 to $number\ of\ layers$ do
 3:     for $k \in C_t \backslash \{c\}$ do
 4:         Call Agorithm 3 to compute layer-wise local sequential influence $\sigma_{(j)}^k$
 5:         if $e_j^k = 1$ then                                   ▷ LWET for non-iid settings
 6:             if $\|\sigma_{(j)}^k\| > \|\epsilon_{t-1,(j)}^{-c}\|$ then
 7:                 $e_j^k \leftarrow 0$
 8:             else
 9:                 $\sigma_{(j)} \leftarrow \frac{n_k}{N(C_t \backslash \{c\})} \sigma_{(j)}^k + \sigma_{(j)}$
10:             end if
11:         end if
12:     end for
13:     $\epsilon_{t,(j)}^{-c} \leftarrow \sigma_{(j)} + \mathbf{w}_{t,(j)} (C_t \longrightarrow C_t \backslash \{c\}) - \mathbf{w}_{t,(j)}$
14: end for
```

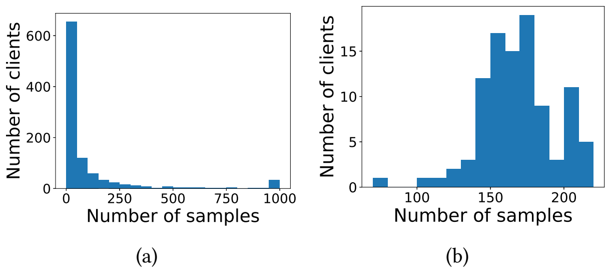


Fig. 7. The statistics of data distributions in (a) setting 1 and (b) setting 3.

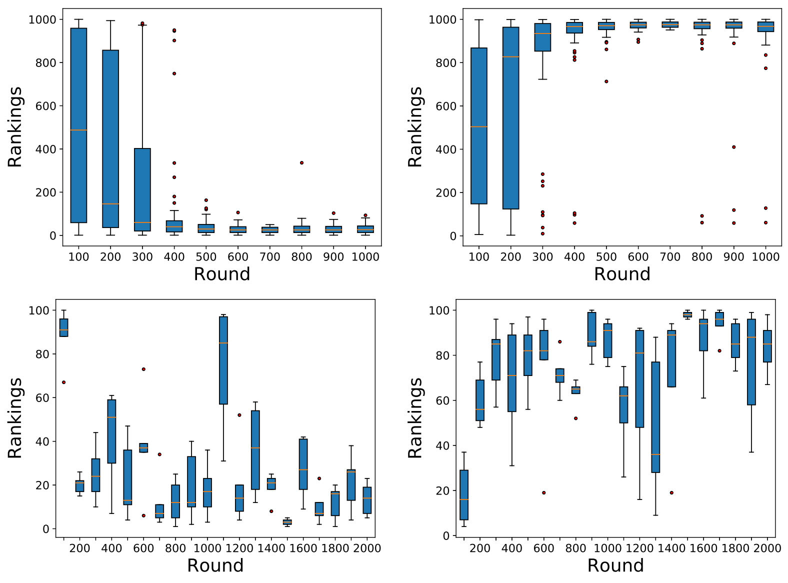
padding="same"]⟶[MaxPool2D, size=2×2, strides=2]⟶[Dense, units=64]⟶[Dense, units=62] ⟶[Output, $y \in \mathbb{R}^{62\times1}$].

- CNN2 has a complete structure: [Input, $x \in \mathbb{R}^{28\times28\times1}$]⟶[Conv2D, size=5×5×4, padding="same"] ⟶[ReLu]⟶[MaxPool2D, size=2×2, strides=2]⟶[Conv2D, size=5×5×16, padding="same"]⟶ [ReLu]⟶[MaxPool2D, size=2×2, strides=2]⟶[Dense, units=128]⟶[ReLu]⟶[Dense, units=62] ⟶[Output, $y \in \mathbb{R}^{62\times1}$].

We use `tf.nn.softmax_cross_entropy_with_logits()` as the loss function.

E TRENDS IN CLIENTS' FIL-BASED RANKINGS

We select 5% of the clients with the highest/lowest FIL at a certain round t, and trace their FIL-based rankings throughout the whole training process. We give the statistics of their rankings every 100 rounds. In setting 1 $t = 700$ and in setting 3 $t = 1500$. Results are shown in Fig. 8. We can see



(a) (b)

Fig. 8. Rankings of (a) the top-5% clients and (b) the bottom-5% clients. Experiments are conducted in setting 1 (shown on top) and setting 3 (shown on bottom).

that these clients' rankings are likely to become more and more concentrated and remain in the top/bottom range as the training progresses.